The Product Attribute Database (PAD): First of A New Class of Productivity Tools for Product Development

Joel C. Sercel
Principal Engineer
Jet Propulsion Laboratory
California Institute of Technology
Pasadena CA 91109

sercel@earthlink.net
www.icsassociates.com
(818) 354-4044

Thomas F. Clymer
Associate Engineer
Jet Propulsion Laboratory
California Institute of Technology
Pasadena CA 91109

tclymer@pop.jpl.nasa.gov
(818) 354-8439

William M. Heinrichs
Associate Engineer
Jet Propulsion Laboratory
California Institute of Technology
Pasadena CA 91109

wmh@mail1.jpl.nasa.gov
(818) 354-3559

Abstract

The Product Attributes Database (PAD) is a new type of information system that serves as a single, network-accessible repository of all the essential parametric information that describes a project at any phase of implementation. PAD goes beyond previous engineering databases in that it includes both a human interface and a machine interface. The machine interface allows a variety of engineering and administrative tools to work together through the database over a network.

The PAD information architecture includes interfaces to several systems including trades models, high-end engineering design and analysis tools, cost estimation tools, design test and verification systems, documentation tools, and data display to support integrated concurrent engineering. The human interface is through an application called the Product Attributes Conversation Tool (PACT) which includes a simple Graphical User Interface (GUI) that allows project team members to read and write to the PAD from any network accessible computer. PACT allows project team members to quickly access PAD data remotely for individual use supporting design and analysis or for group use to support real time collaborative engineering.

The PAD's automated machine interfaces include a range of tools. At the low end of the range are "apparently" simple Publish and Subscribe "sheets" that can be placed into a workbook to allow the user to link spreadsheet calculations to the database. At the high end of the automated machine interface is an API that allows advanced engineering design and analysis tools to link directly to the database over a network.

The PAD database schema was developed concurrently with a naming convention and a security system to allow users to know exactly where to look for a specific design parameter and how to name a new parameter that is to be added to the database. The system is arranged on the basis of Products, Attributes, and States to form Parameters, and an arbitrary number of Versions for each Parameter. A project's database is organized according to its Product Breakdown Structure (PBS). PAD has been beta tested and is in initial use phases with a few flight projects at JPL.

TABLE OF CONTENTS

1. INTRODUCTION

The Need

The recent development of advanced software applications for engineering design and analysis has produced *domain-specific productivity solutions* that can increase the productivity of individuals or small groups working within technical disciplines. Mechanical Computer Aided Design (CAD) and Computer Aided Engineering (CAE) tools are examples of domain-specific productivity solutions and are analogous to similar tools in the fields of electrical engineering, mission design, software development, and system engineering. Transfer of data, specifications, and design changes between tools in different domains is typically slow, labor intensive, and costly. Unfortunately, as product architectures become increasingly integrated, the transfer of data between applications designed for use in different domains becomes more important. As development cycle times shorten, the rate at which data transfer must be accomplished increases. Given these trends, domain-isolated application software can actually reduce worker productivity.

Organizations faced with this dilemma usually apply one or more of several possible solutions in various combinations. One approach is to capitalize on recent developments in

Application Program Interface (API) and object linking technology including OLE (Object Linking and Embedding) and CORBA (Common Object Request Broker Applications). These standards have made it possible to develop individual customized tools that link applications from disparate disciplines. Another approach is to include only tools that use data files based on standards such as IGIS, DXF, or STEP. Most recently, several commercial product development environments have been attempted which integrate a pre-selected set of tools into a unified development environment by capitalizing on a combination of file standards and interface technologies.

Each of these approaches has significant merit and is appropriate to implement in some settings. However, none are without problems. For example, developing a custom API that allows two tools to work together is a fine solution in many cases. However, a typical aerospace project requires tools from 10 to 15 disciplines to work together, and sometimes each of these disciplines requires two or three tools. If a custom API is required between each pair-wise combination of some number n of tools the total number of APIs that must be developed goes roughly as $n(n-1)\approx n^2$. Realistically, to completely integrate the design and analysis tools for a typical aerospace company or laboratory this implies the development of over a hundred pair-wise sets of APIs. For this reason, no organization that we know of has come close to fully integrating their tool sets. For those leaders in the fields that have integrated a fair number of tools into closely related groups, the investment required has been large.

A practical lesson learned from some of these activities has been that once a set of tools has been integrated through API's, tool maintenance becomes a serious concern. For example, it becomes difficult to upgrade to new releases of the various tools because the user organization must upgrade and debug the API to handle any changes before allowing users to utilize the new version. Data format standards are also a vital part of the solution to the tool integration problem and will become more important in the future as standards become more universally adopted, comprehensive, and robust. For now, and into the near future, practical experience suggests that in most cases the needs of tools in different discipline areas are different enough that some type of custom wrapper or file translator is or will be required to adapt the outputs of one tool to the needs of another, even when the use of standards is attempted.

For some companies, especially those with products, processes, and tools that are typical of others in their industry, bundled third party integration environments can be a good practical solution to these problems. However, the practical challenge faced with the use of many third party integration environments include many of the maintenance issues associated with custom API sets, except that the APIs are now out of the control of the user organization and the user organization becomes fatally dependant on the integrating supplier. Fees can start low and then climb. If the supplier of the integrating environment does not support a given tool that your organization needs, you are out of luck. Alternatively, if the integrating environment includes tools to allow custom integration, the use of these tools typically increases licensing fees and/or comes with significant training and learning curve issues for in-house developers. In short, vendor supplied integration environments force you to use the tool suite they provide or pay, one way or another, for custom integration.

The cost of developing interfaces between tools using any of these approaches is frequently found to be high. All too often the result is that organizations still turn to the user-engineers and require them to either manually edit files for translation between applications, or to recreate the same design information in multiple tools. This approach is particularly worrisome because it introduces tremendous opportunities for human error and it makes design configuration management problematical. It also makes the product development process brittle even to small design or requirements changes that are inevitable in the real world.

The Approach

The topic of this paper, the Product Attributes Database (PAD), is an attempt to find a solution to this problem that is inexpensive to implement yet provides many of the advantages of the other integration approaches. While the PAD is not a panacea for these issues, it does go a long way toward ameliorating serious problems. The PAD also provides a powerful interface to the most important component of any development project, the people. It does this in part by collecting and organizing project data into a human readable format that any member of the team can access and link to documents, reports, and presentations.

The PAD system consists of several components. First, the PAD itself is a relational database that has been developed in the Oracle Relational Database engine. Second, interface between the PAD database and most high-end engineering and documentation tools is done through a single C-Language Application Program Interface (C-API). Third, the PAD system includes a very simple Graphical User Interface (GUI) application called the Product Attribute Conversation Tool (PACT) that allows individual users and teams doing real time collaborative design to query the database and obtain human-readable product and development information with minimal training and no knowledge of the underlying database technology. Fourth, the PAD system includes interfaces to standard desktop applications through spreadsheets that allow the unsophisticated user to link PAD data to-from any spreadsheet, presentation, or document that can read and write to Microsoft Office. Finally, the PAD system contains a training program covering best practices and naming conventions that teaches users how to effectively organize, store, and retrieve data.

This Paper

This paper provides a summary description of the PAD system starting with this introductory material in Section 1. Next, Section 2 provides a summary overview of the PAD's role in a project information system showing its interfaces to different tools and systems as it is being implemented at JPL. Section 3 describes the best practices guidelines and recommendations that the PAD team has developed based on a literature review, and more importantly on lessons learned in pilot activities with users projects. The database schema and security systems are outlined in Section 4. The API that links the PAD to other tools is described in Section 5. The specific example of spreadsheet tools is described in more detail in Section 6. Finally, the status of and plans for the PAD system are outlined in Section 7.

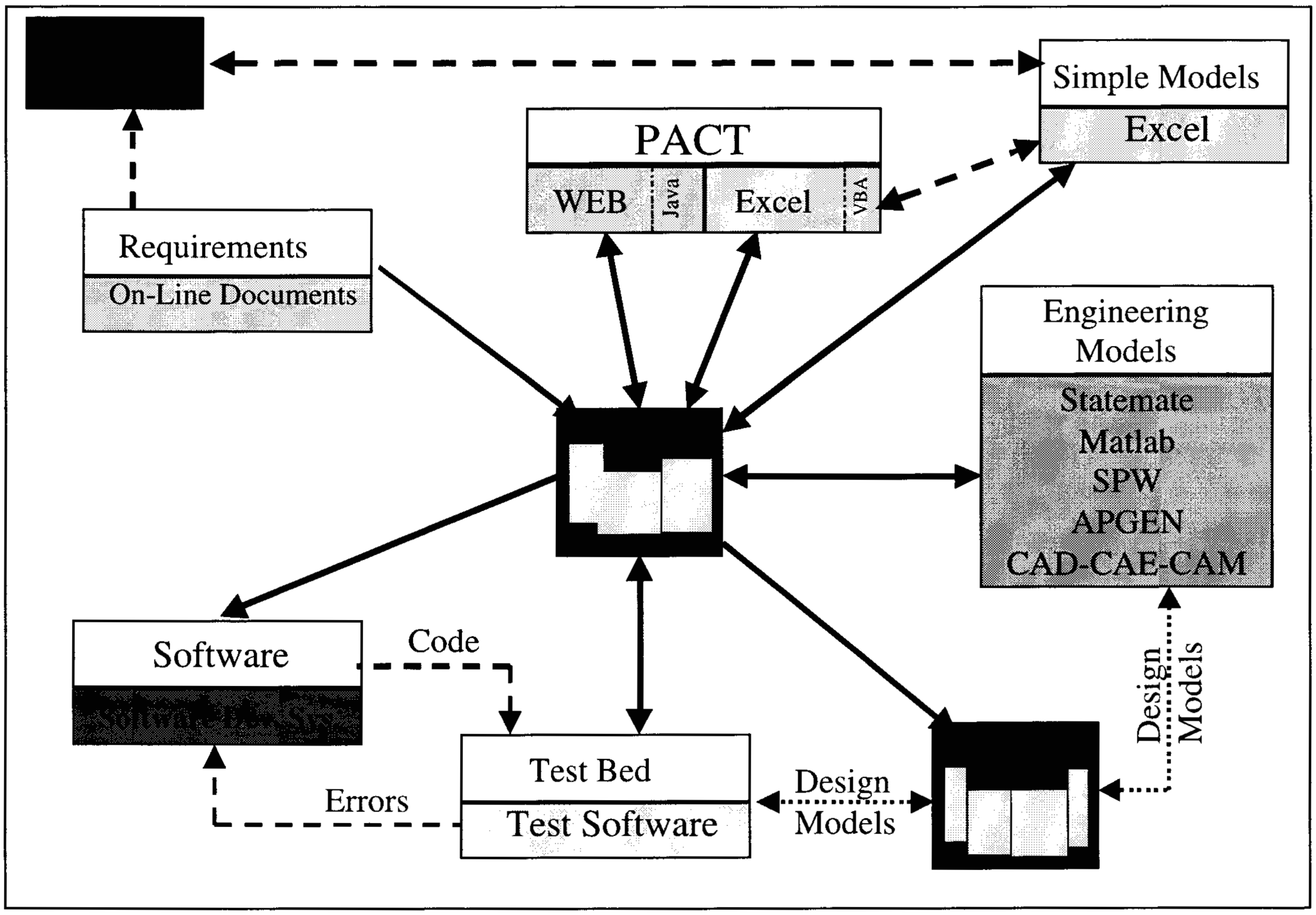
2. PAD Role in a Project Information System

The PAD system has been designed to be the major integrating tool in the information system that supports a product development project. This role is depicted in Figure 1, which shows a simplified schematic of the PAD's interface to other information systems. As Figure 1 shows, the PAD database talks to several classes of tools including the PACT, requirements documentation systems, simple systems engineering and accounting tools, high-end engineering models, a Project Data Management System (PDMS), a Flight Systems Testbed, and a Mission Data System (MDS) or software development environment.

Our discussion of these interfaces will start at the top of Figure 1 with the PACT. The PACT, which is described in more detail in Section 6 of this paper, is a productivity application that allows users to read and write to and from the database and browse and search the database in many ways. We have also recently initiated a Java-based version of the PACT that will support both a stand-alone application, and a web-based implementation. This Java development is scheduled to be in beta test this spring for release in the fall.

We have found that many engineers, administrative support personnel, and managers use or develop their own simple models expressed in many different types of spread sheets in many ways. A major area of productivity loss for aerospace projects is the time spent hunting for the information that is used in these simple models and the distribution of the results. The PAD system addresses this issue through an uncomplicated system of "publish and subscribe" spreadsheets, which are depicted to the right of the PACT in Figure 1. These spreadsheets, which contain code that uses Open Database Connectivity (ODBC) drivers and Structured Query Language (SQL) that is transparent to the user, allows unsophisticated users to link any variable in any Excel workbook to any data in a PAD that they have the appropriate read-write authority for.

The Publish and Subscribe spreadsheets are available to users either as a standalone workbook for integration with new spreadsheets, or as an Excel Add-In that adds the PAD interface to their existing Excel tool menu. Two particular



models have been integrated using the Add-In, the JPL Project Cost Model, and the Outer Planet Solar Probe (OP-SP) Project Trades Model. To give an idea of how much work is involved in integrating a tool with a PAD, JPL Parametric Cost Model integration, which involved reading about 50 technical parameters from the PAD and writing several new cost parameters back into the PAD, took about an hour. The OP-SP PTM, which has a few hundred parameters, took an undergraduate engineering student working as an intern about two days to integrate. This PTM, which is typical of trades models used for rapid exploration of a project's design space, now gives the project system engineer the ability to do what-if studies either from the privacy of his/her office, or in a group setting with a technical team. The tool allows the engineer to access the best and latest design information produced by team members wherever they are on the network. Results can be immediately relayed to the team through the PAD.

Another application of the PAD system is the integration of detailed engineering tools through a single interface rather than requiring the developing of n-squared APIs. As described in a bit more detail in Section 5, this is accomplished by our C-API, which is a very simple interface that has been successfully integrated with every high-end tool we have attempted to work with. The C-API is easy to integrate with high-end tools because it does not require a complex file transfers or sophisticated queries. Rather, the user of the high end tool also takes advantage of the PACT to find out what they need and where their data goes in the database, then they can implement a simple parameter read and write via a publish and subscribe paradigm.

The publish and subscribe paradigm is an important part of the PAD use scenario and is implemented in the interface to Microsoft Office through the publish and subscribe sheets, and to the high-end tools through the API. The concept is that users select groups of data they need to run their tool to get their job done. At the beginning of a design or analysis session they push a "subscribe" button which pulls up the latest version of all their requested data along with meta-data that includes information on who created the information, references to the project library for more background information, and caveats and limitations on how the data should be used. They then run their tool using those parts of the data set that they deem useful. Only when they have developed data needed by others do they push the publish button, which asks them to include the meta-data that places their work in the context that others will need to apply the results.

Publish and subscribe interfaces have been built between the PAD and several high end tools as listed in the diagram. An important tool for JPL is the Matlab, which is used extensively in JPL optics system and interferometer development projects using a Matlab-specific tool called the Integrated Model for Optical Systems (IMOS). In addition to Matlab and the other listed tools we are also building interfaces to tools from other disciplines including for example mechanical CAD and CAE systems.

While the PAD is not a formal design configuration management or Project Data Management System (PDMS), it does utilize such systems and works with them. At JPL the implementation of formal design configuration management is through a PDMS system that is based on the Sherpa application suite. The interface between the PAD and PDMS works through a utility that archives a version history of all PAD data that is web-accessible in PDMS. The utility also makes it possible to automate the process of establishing a project PDMS system by populating a Product Breakdown Structure (PBS) in PDMS automatically from PAD data. The interface to the PDMS system is still in development and we expect to be operational with this utility by the end of May.

PAD system applications are not limited to design and analysis. *As tested* versions of technical parameters that are produced by in facilities such as JPL's Flight System Testbed are also being integrated with the PAD through the C-API using the publish and subscribe paradigm. Using this approach, a test engineer can read test requirements directly from the network to his test station and can then conduct tests. Results of tests can then be read back into the database at near real time giving cognizant engineers extremely rapid and timely reports of test results. This capability is presently available and is being tested in a pilot study.

The interface to the software development system is still very early in its development cycle. The plan is to make it possible for software developers who are writing software that will be resident on flight hardware or which is critically dependant on flight hardware to be able to subscribe to the relevant technical parameters as part of the routine process of software development. One envisioned application is the development of a utility that subscribes to technical parameters that are needed for .h files at software build or compile time. This utility would obtain the most up-to-date approved values of the various parameters and look for changes that are required in the build.

The PAD system has several interfaces to project documentation. The previously mentioned link to spread sheets makes it possible to tie any technical information in any project document to the project database for machine assisted updates. More powerful documentation tools such as DOORS, which JPL uses to track requirements flow on several projects have also been integrated with the PAD. Another important feature of a project information system is a project library. Documents tied to the PAD can be stored and retrieved by project personnel through a web-based project library using any one of a number of COTS technologies. Xerox Docushare is an example of a system that we have tested and proven to be compatible with PAD technology.

3. PAD ORGANIZATION AND BEST PRACTICES

The aerospace industry has a long history of bad experiences with database information systems. Frequently these systems take more work to maintain then they save and some have become so cumbersome that they are abandoned even after the investment of millions of dollars. Our experience is that this can happen for many reasons including a failure of the developers to understand the actual engineering process or a failure of the users to apply best practices in the use of the database. To ensure that the PAD tools are effective, we have developed them in close collaboration with actual space project development teams. We have applied an incremental test-build-test-build philosophy where we do not add another layer of functionality until we have determined that the existing system does what it needs to do for the users.

In this process we have found that untrained users or users who ignore best practices can populate the database in such a way that they can not find what they need, or can not use what they find. The only way around the garbage-in-garbage-out factor is to ensure that the data that goes into the system is as "good" as possible and that processes are in place for continually monitoring and improving the extant data. A requirement for ensuring that a minimum of bad data goes into a database is that the project personnel know how the PAD data is organized and structured and that they are aware of the best practices. An overview summary of this information is presented here to communicate a sense of what is involved.

The first aspect of best practices is determining what should and should not go into a PAD. It is actually more important to know what should not go into a PAD because too much extraneous information can make the system confusing and difficult to use. The following classes of information are not suitable for inclusion in a PAD system:

- Information that only one member of a project team needs or uses.
- Information that is not best kept in an atomic parameter format, e.g.:
- Bit streams
- Data files
- Documents

It is natural to ask what should go into a PAD. The answer includes:

- Parametric technical data
- Data that is needed by more than one person
- Data that effects the design of the product
- Data that can be read into or out of a tool using the C-API
- Data that fits the PAD naming conventions

To determine if a given set of data fits the PAD naming conventions, it is necessary to understand that the logical organization of PAD data starts with a Product Breakdown Structure (PBS) and includes the definition of Parameters, Parameter Versions and team member roles.

Product Breakdown Structure (PBS)

The PAD is called the *Product Attribute Database* because we have found that all technical and administrative data important to the development of a product or the completion of a project can be thought of as attributes of products or sub-products of the system being developed or delivered. The first challenge is to find a way to organize the products into a hierarchical set that decomposes the primary product into groups of products each of which can be further broken down to lower and lower levels. This hierarchical, tree-type organization of products is called a *Product Breakdown Structure* or PBS. A PBS is a tree-type structure in that each product in the tree is the sum of a set of products that we refer to as its *children*. For the purposes of a PAD there can be as many levels of children as are needed to fully describe a product structure, though even for complex space project more than six or seven levels is typically not needed. It is interesting and useful to note that a PBS can be defined for a Program, a Project, a System, or any element thereof. As such, a PBS for a subsystem can be thought of as a complete PBS although it is only a component of a larger system PBS in a larger project structure that may be part of a program.

Before building a PBS it is necessary to have at least a rudimentary functional architecture of the system and an idea of what a product is. The development of the functional architecture of the system is the role of system engineering and is not addressed in this paper. We have found that it is useful to think of products as any hardware, software, documentation, presentation, or mission operational event that is accomplished by the Project Team. Treating mission operational events as products is difficult and requires subtle distinctions. The types of events that can be valid products do not include the completion of developmental milestones, delivery of products, or the completion of a project phases. An event is only a product if the completion of the event per-se constitutes a product. A specific maneuver that is detailed in a requirement on a test flight in an aircraft development and a trajectory correction maneuver in a space project are both operational events that are valid products. It is necessary to treat mission operational events as products because certain technical parameters are attributes only of events and would otherwise have no valid place in a PAD. For example, a trajectory correction maneuver has delta-V as an attribute and a test flight has a maximum rate of climb as an attribute. Both of these attributes are engineering data that make more sense to associate with operational events than with hardware or software.

We have found in using a PBS to organize information describing attributes of products that many of the same best practices that are found in the literature for how to build and maintain a PBS for a project using paper systems are still applicable. In addition, the first few levels of a space project PBS look like a product oriented Work Breakdown

Structure (WBS). A few of the best practices or guidelines we have found useful in building a PBS for a project database system are listed below:

- Ensure that every product is a product:
    - Services, support, and project phases are <u>not</u> products
    - Management and system engineering can be expresses as products. Management produces products like implementation plans and reports. System engineering produces produce interface specifications and requirements.

- Use Specific (Not Generic) Product Names
    - e.g., *Battery Control Electronics*, Not *Electronics*
- Ensure that the children of a product form the parent when taken together
    - If something else is needed to form the parent, the PBS is wrong
- Don't confuse your PBS with your document tree or your org-chart
    - But be clear about the relationships between them
    - A PBS may include documents which may be legitimate products
    - A PBS may be closely related to the organization chart
- Products from support activities including management, system engineering, rework, retest, and refurbishment should be treated as part of the appropriate product (a child of), not as a stand alone hierarchy
- Non-recurring and recurring classifications are not separate products, but may be separate attributes
- Attributes and values are not specified in product names
    - Instead of using .9N Thruster, use Thruster
- Software that is developed to reside on specific equipment should be part of (a child of) that equipment in the PBS

Attribute

The primary purpose of the PBS in the PAD system is to provide a product structure to which technical and administrative data can be tied. Any technical or administrative data that one might want to tie to a PAD can be though of as *attributes* of the products in the PAD. An attribute is a measurable description of a product. Attributes can be variable, like power consumption, or they can be constant, like net dry mass. In addition, attributes can be state dependant or independent of state. For example, mass does not change when you turn something on or off, but power consumption does. Most attributes are numerical in nature and they have units (Voltage (v), Mass (kg), and Power (W) are examples), but attributes don't have to be numerical. For example, material type is an example of an attribute that is not numerical.

As with products it is important to use best practices and conventions in working with attributes in a PAD. The first area this applies to is in naming attributes. An important rule here is to use specific terms including units, if applicable, in place of generic terms that can mean many things. For example, *Power* is not an acceptable attribute name, but *Total Power Consumed (W)* and *Power Consumed at 28 Volts (W)* both might be appropriate attributes of a piece of electronics. Similarly, the term *Energy* is not a good attribute to describe a battery because there are many different energies associated with a battery. *Total Battery Capacity (J)* and *Battery Charge State (J)* would be better attributes of a battery system.

We have found that in assigning attributes it is not always obvious which product an attribute belongs to. For some attributes, like mass, it is easy. For others, like temperature, untrained personnel tend to want to put the attributes in different places. For example, some people want to assign temperature to the thermal control system, while others want to assign it to the item whose temperature is being controlled. This confusion is a problem because it can make it difficult to know where in the database to get the information one seeks. We have found, empirically, that the system works best if control attributes are attributes of the product being controlled, not the control system. This way users always know where to find attributes of a product. Under this convention, peak temperature is not an attribute of a thermal control system, but is an attribute of a battery, an electronics box, a slice, or a propellant tank. Likewise, pointing jitter is not an attribute of an ACS system, but it is an attribute of a platform controlled by an ACS system. Pointing knowledge would however be an attribute of the ACS system.

State

As mentioned previously, attributes can be state dependent. As such, it becomes necessary to track the states of a product as well at its attributes. For the PAD, a state is an operational condition or mode that effects the value associated with an attribute of a product. In general, different products have different sets of states. For example, valves are typically either “open” or “closed” but electronics are typically “on”, “off”, or “standby”, and can have many other important states. Defining product states is required to track how attributes of a product change with operation. For attributes that don't change with state, typically the dry mass of a bolt does not change with operation, the "All" state is used.

In using states it is important to keep in mind that specific events are not states. For example, *TCM-27* (trajectory correction maneuver number 27) is not a state, but *Thruster Firing* might be. Likewise, project phases are generally not states, but they might be Value Names (see below) or Attributes. A few reasonable examples of states are BOL or EOL for beginning or end of life, Launch, Cruise, On, Off, Standby, Warm-up, Beginning of Blowdown, and End of

Blowdown. An attribute that has been assigned a product and a state forms a *Parameter.* A few examples of parameters are listed in Table 1.

Table 1. A Few Simple Examples of PAD Parameters

Product	Attribute	State
Solenoid Valve	Power Consumption (W)	Open
Star Tracker	Data Output Rate (bps)	Calibration Mode
Battery	Max Allowable Temperature (K)	All

Value Name and Versions

If human beings and the products they build were perfect, the logical structure of the PAD could end with parameters, and numbers or other types of values could be kept at the parameter level in the database. Unfortunately, there are many different versions of any given *number* a product development team must track. To help, we have created the *Value Name* field in the PAD to provide a simple designator or name of the different versions of PAD parameters. Examples of different Value Names or Versions of a given parameter might be *Vendor Specification*, *Subsystem Requirements Document*, *CBE* (for Current Best Estimate), *Allocated*, *Max Tested*, *Min Tested*, and *Project Official Version.* Note that in most cases the Value Name indicates the source of the data.

In the practical use of the PAD technology in our pilot projects we have found that one group of parameter versions stands apart from the rest in terms of how the project team uses them. The group in question is the requirements versions. To make it possible for requirements to be treated differently in a PAD we have created another database field called the *Requirement Type.* The Requirement Type is a flag the PAD Database uses to identify requirements and make it possible to search on a subset of possible requirements violations. In addition the Requirement Type filed can make it possible for contents of a requirement parameter to be written into the database only from the official project documentation, which typically exists not in paper form, but in a document system such as DOORS.

Table 2. Example Showing PAD Fields and Corresponding Entries for a Single PAD Parameter Version

PAD Field	Example Content
Product	Power System
Attribute	Use of Pu
State	All
Value Name	Project Requirement Doc.
Requirement Type	Nom
Value	True

If a Parameter Version is a requirement, the Requirement Type field can be set to Max, Min, or Nom (for Nominal). Quantifiable requirements are typically Max or Min, while qualitative requirements are typically Nom. For example, if a space project has a requirement not to use a plutonium power source, as all Discovery missions do, there might be a parameter version with the database fields in PAD populated as shown in Table 2. This example is actually a completed version of a parameter, or a *Parameter Version.* A Parameter Version is a unique combination of Parameter, Value Name, and Requirement Type. Table 3 provides a few more examples

Note that in this table the last two Parameter Versions are versions of the same Parameter, namely the ACS Thruster Temperature in the On State. One of the versions is the maximum temperature as specified in the requirements document, the other is the minimum temperature. These requirements are placed on the thermal control system in the thermal control system requirements document and linked to the PAD at publish time using the C-API. Once Parameter

Table 3. A Few Examples of Parameter Versions That Might be Found in a Project PAD

Parameters				
Parameter Versions				
Product	Attribute	State	Value Name	Requirement Type
Power System	Use of Pu	All	Project Requirement	Nom
Star Tracker	Output Rate (bps)	On	Max Tested	None
ACS Thruster	Min I-Bit (N•s)	Cold	Propulsion Sys Req.	Min
ACS Thruster	Min I-Bit (N•s)	Cold	Max Tested	None
ACS Thruster	Min I-Bit (N•s)	Cold	Max Tested	None
ACS Thruster	Specific Impulse (s)	On	Propulsion Sys Req.	Min
ACS Thruster	Specific Implies (s)	Cold	Propulsion Sys Req.	Min
ACS Thruster	Temperature (K)	On	Thermal Control Req.	Max
ACS Thruster	Temperature (K)	On	Thermal Control Req.	Min

Versions have been defined they can be populated with data. The data that a Parameter Version holds is referred to as the *Value* of the Parameter Version. Values are typically numbers and can be up to 100 characters.

Product Owners and Value Owners

The database recognizes two types of people who can enter data into a PAD: *Product Owners* and *Value Owners*. A Product Owner is a person who is responsible for (has control of) a product. The owner of a product is also an owner of all of all the children of the product, and their children. By recursion this ownership extends all the way down throughout the PBS. For example, a spacecraft may be composed of several systems: a science system, a propulsion system, a telecommunication system, and others. The Product Owner of the spacecraft would also be the Product Owner of the science system, propulsion system, and the others although each of the others should also have its own individual Product Owner.

Product Owners have the following roles the development and use of a PAD:

- Creating, editing, and deleting products.
- Assigning parameters and parameter versions to their products.
- Assigning Product Ownership to lower level products
- Assigning Value Owners to Parameter Versions.

While Product Ownership authority extends down through the PBS, the recommended practice is to have the lowest level Product Owner who has the authority to act in the database be the person who actually executes database functions. This way, high level Product Owners are not overwhelmed with the amount of data they are responsible for developing and maintaining.

Another type of person recognized by the database is the *Value Owner*. Value owners are the people responsible for numerical or other type of the value and description of the corresponding parameter version. Examples of Value Owners might include a system engineer who will take ownership of the requirements versions, a test engineer who may own versions related to testing, and the Cognizant Engineers (CogE's) who may own current best estimates. The Value Owners put the actual values of parameters into the database.

Description Fields

Several description fields are available in the database where the people who populate the database can write descriptions of the data to provide data users the information they need to make effective use of the data. The most important of the description fields are the Product Description, Parameter Description, and the Value Description. Product Descriptions are the first step in developing an on-line data dictionary for a project and are analogous to a Work Breakdown Structure (WBS) dictionary. The Product Description should be a sentence or

Figure 2 The PAD Database Schema

so of text that identifies the product to the extent that it is non-obvious from its name. Likewise, the Parameter Description records the technical meaning of each parameter for Value Owners and other users. The parameter description should tell users anything that is non-obvious from the product name, product description, attribute name, and state name. Parameter Descriptions most often focuses on clarifying exactly what the attribute means. Value Owners use the Value Description field to communicate how a number was produced, how reliable it is, and any limitations that should be placed on its use. A general warning for PAD users is to read the Value Description before using any value found in the database. Value Descriptions typically include assessments of how accurate a number is, when the Value Owner plans to update it, and a reference to more information if it is needed.

4. Database Schema and Security

The PAD database schema and security features were developed to support the naming convention and logical structure outlined in Section 4 of this paper.

The schema, which is shown in Figure 2, contains 12 tables and corresponds closely to the naming conventions discussed earlier. The def_products, def_states, def_attributes, and def_value_names tables all contain the user defined lists of products, states, attributes, and value names. Requirement types do not require a table to store their definitions, as this dimension of the data is restricted to only four values, "None", "Min", "Nom", and "Max", and is not user-modifiable. The product_comp table records the parent-child relationships between products; a product that appears as a child more than once in this table appears more than once in the PBS, and these instances of the product are considered "linked". The def_pbs_levels table records the names of each level of the product breakdown structure. The def_params table records the defined combinations of products, attributes, and states to form parameters, while the def_param_vers table records the defined combinations of parameters, value names, and requirement types (for historical reasons requirements type designators are stored in a field called "value type" in the schema) to form parameter versions. The staff table contains the project-defined list of users who can be either product or value owners, as well as contact information on each.

The temp_product_hierarchy table circumvents a constraint on queries within Oracle SQL preventing the combination of a PBS navigating query with table joins. The temp_product_hierarchy table provides a location for users to store an organized PBS (generated through a stored procedure), after which they can execute queries joining the temp_product_hierarchy table with the other tables in the database.

The triggers in the PAD perform a variety of functions, primarily clerical and security. The clerical functions include generating a new ID number for a record based upon a sequence, recording the username of a user that manipulates the data, and recording the date/time at which the data was altered. The security functions are concerned primarily with enforcing the rules associated with product and value owners. Product Owners are able to create sub-products, parameters, and parameter versions at any level beneath a product they own, as well as being able to define and modify attribute, state, and value name definitions. As a part of defining a sub-product or parameter version, a Product Owner may assign a sub-Product Owner or a value owner. Value owners are allowed to alter the value of an existing parameter version for which they are the Value Owner.

5. C-Language Application Programming Interface (C-API)

The PAD C language, Application Programming Interface (C-API) enables high performance engineering tools access to technical parametric data through a standardized suite of function calls. Direct PAD access reduces the possibility of erroneous design tool output due to human data input errors. Use of the C-API enhances productivity by programming the

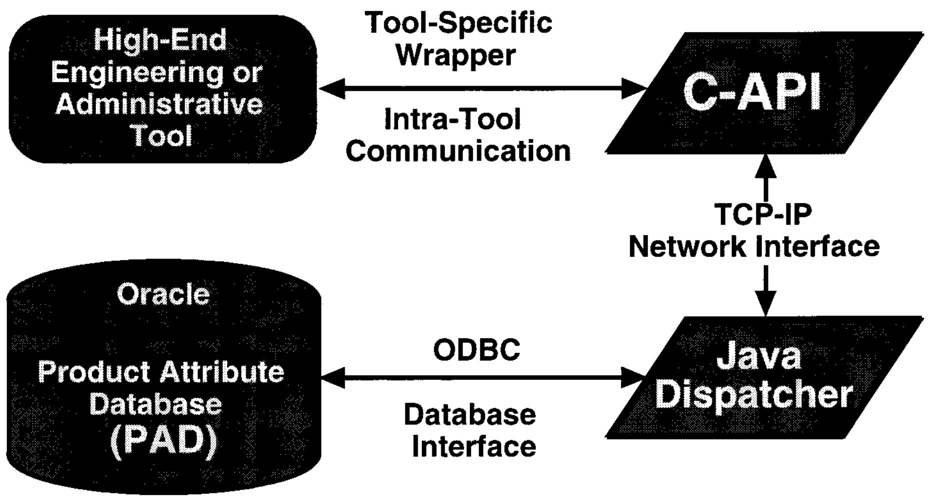


Figure 3. A Simplified Diagram Showing the Interface Between Design Tools and the PAD

data population and storage into the design tool, thus removing the chores of locating the most-recent data and archiving results.

Our investigation into popular engineering design tools discovered that many do not provide any database access through ODBC (open database connectivity). ODBC is an industry standard protocol for access to commercial database engines. Fortunately many tools were written in the C language and provide access to user-developed C code. With this in mind, the PAD C-API acts as a wrapper around ODBC giving engineering tools the benefits of database connectivity.

The C-API is a suite of function calls housed either in a library (.lib, .dll, etc.) or as object files used to compile with a design tool (Statemate, Foresight, etc.). The function calls contain structured query language (SQL) that communicates with the PAD. The SQL is passed via TCP/IP networking protocol to a dispatcher daemon running on a Sun workstation. The daemon enacts ODBC calls to the PAD and will package any return data and return it to the client C-API. The C-API parses the page from the dispatcher and returns it to the design tool. An important advantage to this architecture is the ability to run engineering design tools on many different workstations without having to configure ODBC settings on each. Any updates to the PAD schema and corresponding SQL will result in a change to the underlying code of the C-API; the design tool need only update the old library.

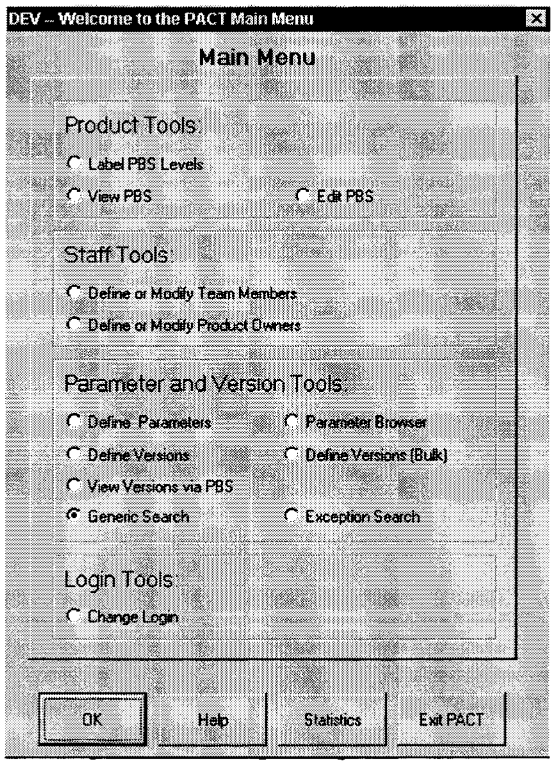


Figure 4 The PACT main menu.

6. Spreadsheet Tools

The PAD system includes several features that are built into common office software including Microsoft's Excel spreadsheet application. The Excel based tools include the Product Attribute Conversation Tool (PACT), Publish and Subscribe spreadsheets, and the Bulk PBS builder. A brief description of each of these tools is provided next.

PACT: The Product Attribute Conversation Tool

PACT is an Excel-based software application designed to support user interaction with the PAD. We envisage that for most users PACT will be the primary tool for searching and modifying a project's PAD. PACT makes it possible for users to access PAD data in a read-only mode or read-write mode.

PACT is a productivity application that allows users to read and write to and from the database and browse and search the database in many ways. The PACT main menu, which lists some of the available features, is shown in Figure 4. PACT uses Excel's Visual Basic for Applications (VBA) language to support rapid deployment of new features requested by users. In addition to the Excel implementation, we have also recently initiated a web-based implementation of the PACT and a stand-alone version, both of which are being developed in Java and are scheduled to be in beta test this spring for release in the fall. The existing VBA version of the PACT application can be viewed as a rapidly developed prototype. In the use of this prototype we have learned about which features are most important, and we are keeping the best for the Java implementation.

PACT incorporates all necessary functionality to manage project parametric data into a single desktop tool. PACT's design rules mandate simplistic user-screens fashioned toward consistency, and recognizable functions. PACT has the ability to create, or modify product breakdown structures (PBS). Products, and their child-tree products can be deleted, created, or moved through a series of mouse clicks. Parameters and parameter versions can be created en-masse through simple drag & drop interfaces. Project staff members can be created in the PAD and have their personnel contact information updated through PACT. Product and parameter version ownership is assigned with mouse clicks.

PACT provides two different searching capabilities: generic and exception. Generic searching outputs parameters or parameter versions for a specified product and, if desired, its children. The parameters can be filtered through a selection attribute, state, value name, requirement type or any combination there-of. A list can be compiled in the results window and later exported to an Excel spreadsheet. Exception searching enacts predefined parameter searches based on Product Ownership, date, or failed requirements.

Typical PACT use scenario's can include group or individual settings. In one application teams can use PACT during meetings to identify failed requirements and discuss their impact upon the mission. Requirements or designs can consequently be modified on the spot. PACT's design rules mandated that usage scenarios would not fully determine its design. Our goal is to provide a base set of functionality and let the user create their own scenarios that best fit their requirements, work-style, etc.

Data can be exported from many different PACT user screens into Excel spreadsheets. This feature is useful in creating reports, populating the publish & subscribe PAD workbooks, and for convenient desktop storage of PAD data.

Publish and Subscribe Workbook

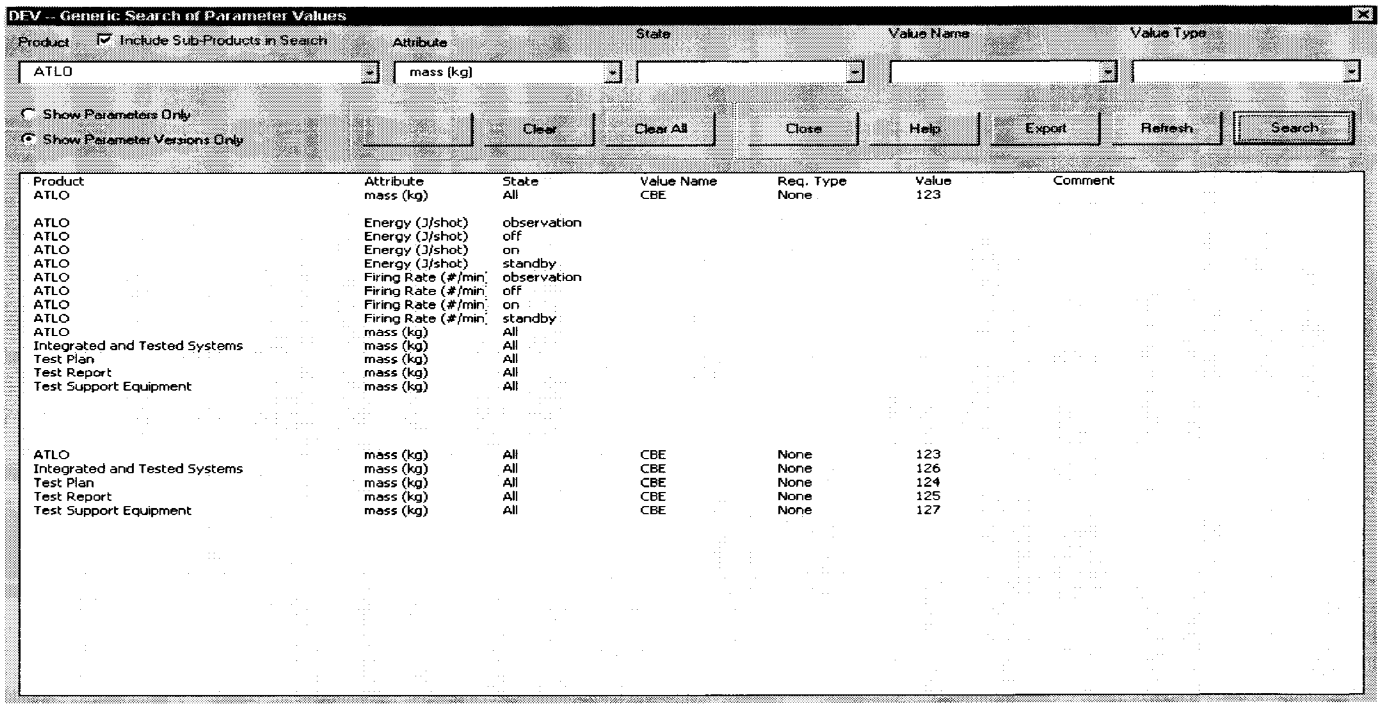
The PAD publish and subscribe workbook enables Microsoft Office tools easy connectivity to PAD data through Excel. The workbook contains two worksheets: *Publish*, and *Subscribe*. On the Publish worksheet, parameters are written or archived to the PAD. The subscribe worksheet allows the user to read from the PAD to spreadsheets on the client machine. Setting up the publish and subscribe parameters is facilitated through PACT's export feature. PACT's generic search functionality allows compilation of desired parameters in a format that models the publish and subscribe columns. Once initiated, the publish and subscribe sheets acquire and archive PAD data with a menu item click.

The publish and subscribe worksheets have many uses, the most common being integration of simple budgeting tools and standardized reports. For example, subscribed mass data is input for mass distribution graphs and margin tables that are linked to PowerPoint slides. Before (or during) a monthly review, the latest data can be subscribed, automatically updating the presentation with the latest mass data in a matter seconds.

The publish and subscribe workbook is also available as an Excel add-in. The add-in provides very simple integration with pre-existing models and reports. The add-in contains all the code for publishing and subscribing; in addition, it generates (if needed) the publish and subscribe worksheets.

The Bulk PBS builder

There are two ways to create products in PAD. Depending upon where the projects life cycle is will dictate which way is best for you. The first and probably most common method is to create a product inside a PAD where the product breakdown structure is currently defined. The second is to create products and product breakdown structures simultaneously. The latter is usually used during PAD initialization, or during large expansions of the product tree. The PBS Builder can be used to initialize a PAD or add an extended branch into an existing PBS. The PBS Builder is intended to allow the user to enter a large PBS into a PAD very quickly, rather than entering the products individually through the PACT.



Product [✓] Include Sub-Products in Search: ATLO
Attribute: mass (kg)
State:
Value Name:
Value Type:

( ) Show Parameters Only
(●) Show Parameter Versions Only

Product	Attribute	State	Value Name	Req. Type	Value	Comment
ATLO	mass (kg)	All	CBE	None	123	
ATLO	Energy (J/shot)	observation				
ATLO	Energy (J/shot)	off				
ATLO	Energy (J/shot)	on				
ATLO	Energy (J/shot)	standby				
ATLO	Firing Rate (#/min	observation				
ATLO	Firing Rate (#/min	off				
ATLO	Firing Rate (#/min	on				
ATLO	Firing Rate (#/min	standby				
ATLO	mass (kg)	All				
Integrated and Tested Systems	mass (kg)	All				
Test Plan	mass (kg)	All				
Test Report	mass (kg)	All				
Test Support Equipment	mass (kg)	All				
ATLO	mass (kg)	All	CBE	None	123	
Integrated and Tested Systems	mass (kg)	All	CBE	None	126	
Test Plan	mass (kg)	All	CBE	None	124	
Test Report	mass (kg)	All	CBE	None	125	
Test Support Equipment	mass (kg)	All	CBE	None	127	

Figure 5 PACT's generic search user-screen.

7. Status and Future Plans

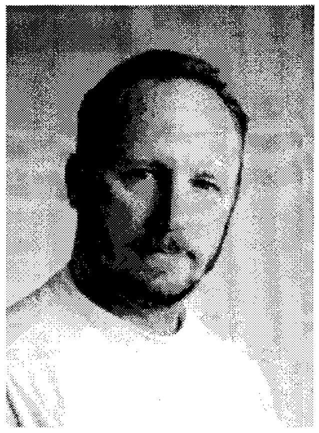
The PAD system is a new approach to integrating information systems to support rapid development of high quality products. While this approach is still in its infancy, some of the lessons learned so far include a set of best practices that have been outlined in this paper and the awareness that an organization should not attempt to deploy a system such as this without pilot activities to develop operational scenarios. In addition, no integrating database such as the PAD should be put into the hands of untrained users. While the information system itself is fairly simple and essential tools such as the PACT and the Publish and Subscribe sheets can easily be learned in an hour or so, the ability to generate and use effective content requires significant training. Our experience is that Product Owners should receive no less than about 6 hours of training including 3 hours of theory and 3 hours of hands on work in the presence of an experienced instructor. Value Owners require about 2 hours of training on the theory and about an hour and a half of hands on training with the tools before the can make the most effective use of the system.
Version 1 of the PAD system has completed beta testing and has been used in small pilot activities with a few projects at JPL. At present we are observing the use of the Version 1 system with those pilot projects as its application grows to determine which features are most helpful and which should be eliminated. This information is being incorporated into the Version 2 system that we hope and expect will be the system that is applied universally to flight projects at JPL.

While JPL is planning on using the PAD system for space projects including flight systems, missions, and instruments, there is nothing about the design of the database or its usage scenario that is unique to space or even aerospace projects. It is our hope that this system or systems like it will be used for all types of complex product development processes to reduce the tedium associated with engineering and administrative functions, reduce errors, improve product quality, and reduce development time and cost. Those interested in learning more about the PAD system should feel free to contact the first author of this paper for additional information as it becomes available.

Acknowledgements

The authors would like to express their appreciation to the rest of the PAD development team for their hard work and intellectual contributions. Mr. Craig Peterson of JPL is acknowledged for his contributions to the development of usage scenarios and product testing in the context of our pilot with the Space Interferometer Mission (SIM) project. Adrian Gody is acknowledged for developing the C-API and associated documentation. Justin Tenicsi and Troy Schmidt are acknowledged for their contributions to developing the PACT user interface and for product testing. The authors would also like to express their appreciation to the Outer Planet Solar Probe (OP-SP), SIM, and Deep Space 4 Project teams for providing product information and testing of the PAD systems and processes.

Biographies



Joel Sercel is a principal engineer at the Jet Propulsion Laboratory and is a Visiting Associate and Lecturer in Aeronautics at the California Institute of Technology. Dr. Sercel has innovated in the areas of spacecraft technology, space systems engineering, and product development processes and tools. In the early 1990's along with Dr. John Brophy of JPL, Sercel conceived of and defined the NSTAR project which has culminated in the flight of the world's first deep space ion propulsion system on the New Millennium Program's Deep Space One Mission. Dr. Sercel is the Director of a laboratory at Caltech which specializes in developing tools and techniques to improve the productivity of space mission and system product development teams and he teaches a graduate level course at Caltech in Space Missions and Systems. Finally, Sercel consults in the area of information system assisted collaborative engineering and teaches short course in space systems engineering and methods of collaborative design and engineering.

Thomas F. Clymer is an Associate Engineer at the Jet Propulsion Laboratory, where he has worked for the past two years, primarily as the primary programmer of the PAD system and related development activities. Mr. Clymer designed the PAD database schema and implemented the database and its triggers. He also wrote most of the code for the PACT user application. Prior to this, Mr. Clymer was a student at Caltech where he received his MS in Aeronautics in 1996.

William M. Heinrichs is an Associate member of the staff in the Jet Propulsion Laboratory's Engineering Economics & Costing group specializing in database projects. His current activities involve coordinating the programming team and collaborating in the design and implementation of the PAD. In addition he is prototyping an inventory-analysis database system for the Deep Space Network. Heinrichs holds a BS in computer science and an MBA both from Rensselaer Polytechnic Institute.

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.